

# **Generically Used Expert Scheduling System**

***Guess***

**User's Guide**  
Version 1.0

**American Minority Engineering Corporation**  
10422 Armory Avenue  
P.O. Box 509  
Kensington, Maryland 20895

**March 31, 1996**

## Table of Contents

<b>1.0 Introduction</b>	<b>1</b>
1.1 Identification of Document	1
1.2 Scope of Document	1
1.3 Purpose and Objectives of Document	1
1.4 Document Status and Schedule	1
1.5 Documentation Organization	2
<b>2.0 Related Documents</b>	<b>2</b>
2.1 Parent Documents	2
2.2 Applicable Documents	2
2.3 Information Documents	3
<b>3.0 Overview</b>	<b>3</b>
3.1 Background	3
3.2 Major Features of GUESS	4
3.2.1 Suggestion Tabulator (Sugtab)	4
3.2.2 Hill Climbing	5
3.2.3 Genetic Algorithm	5
3.2.4 User Interface Design	9
3.2.5 Resource Modeling	10
3.3 Performance of GUESS	13
3.4 Future Direction of GUESS	13
3.4.1 Developing Database Interfaces to GUESS	13
3.4.2 Categorize Different Scheduling Problems and Develop Generic Scheduling Models for Each Within GUESS	14
3.4.3 Develop a Test Suite of Different Types of Scheduling Cases to Run Against GUESS Each Time a New Scheduling Technique is Included in GUESS	14
3.4.4 Improved Methods for Customizing the Report Generation Function of GUESS	15
3.5 Summary	15
3.6 References	15
<b>4.0 Installation</b>	<b>16</b>
<b>5.0 Working with Guess</b>	<b>17</b>
<b>6.0 Using the Guess Scheduling Program</b>	<b>17</b>
6.1 Viewing the Schedule	17
6.2 Event List Pane	18
6.3 Gantt Chart Pane	19
6.4 Adding/Editing Events	20
6.4.1 Editing the Event	21
6.4.2 Editing the Constraints	22

## Guess User's Guide

6.4.3 Editing the Resource Constraints	25
<b>6.5 Resources</b>	<b>25</b>
<b>6.6 Scheduling</b>	<b>26</b>
<b>6.7 Saving Your Work</b>	<b>28</b>
<b>6.8 Printing the results</b>	<b>28</b>
<b>7.0 Abbreviations And Acronyms</b>	<b>30</b>
<b>8.0 Appendices</b>	<b>31</b>
<b>Appendix I</b>	<b>31</b>

## Guess User's Guide

### *Table of Figures*

<i>Figure 1 - Chromosome Encoding</i>	6
<i>Figure 2 - Producing a New Generation</i>	7
<i>Figure 3 - Mutation Rates</i>	8
<i>Figure 4 - Crossover Probabilities</i>	9
<i>Figure 5 - Main Window Menu</i>	17
<i>Figure 6 - File Open Dialog</i>	17
<i>Figure 7 - Schedule Window</i>	18
<i>Figure 8 - List Pane</i>	19
<i>Figure 9 - Gantt Chart Pane</i>	19
<i>Figure 10 - View Menu</i>	20
<i>Figure 11 - Satisfaction Dialog</i>	20
<i>Figure 12 - Event Dialog</i>	21
<i>Figure 13 - Editing Constraints</i>	22
<i>Figure 14 - Entering a Meta Constraint</i>	24
<i>Figure 15 - Editing Resource Constraints</i>	25
<i>Figure 16 - Resource Graph</i>	25
<i>Figure 17 - Resource Menu</i>	26
<i>Figure 18 - Resource Dialog</i>	26
<i>Figure 19 - Scheduling Methods</i>	26
<i>Figure 20 - Statistics Dialog</i>	27
<i>Figure 21 - Resource Conflicts</i>	28
<i>Figure 22 - File Save As Dialog</i>	28
<i>Figure 23 - Print Dialog</i>	29

# **Guess User's Guide**

## **1.0 Introduction**

### **1.1 Identification of Document**

This is the User's Guide for the operation of GUESS (Generically Used Expert Scheduling System) NASA Project NAS5-38062. The purpose of the User's Guide is to provide end users (rather than system operators or administrators) with instructions explaining how to execute the software effectively.

### **1.2 Scope of Document**

GUESS is a generic expert scheduling system. Pinedo [1] suggests important features of a generic scheduler which have been incorporated into GUESS such as:

1. automatic scheduling routines to generate a "first" schedule for the user;
2. user interface that includes Gantt charts and enables the human scheduler to manipulate schedules manually;
3. diagnostic report generators; and
4. a variety of scheduling techniques included in the generic scheduling toolkit.

The GUESS User's Guide contains all the information needed to load, initialize, and execute GUESS on an IBM PC or compatible computer using Windows. This document adheres to the NASA Software Documentation Standard Software Engineering Program Standards (NASA-STD-2100-91) for a User's Guide.

### **1.3 Purpose and Objectives of Document**

The purpose of the document is to provide a well organized, easy to use guide for the user of the GUESS software system. It is intended to guide the user through the steps necessary for installation, start-up, initialization, operation, and termination of the GUESS program. This document should help the user in running GUESS and applying GUESS for solving a scheduling application.

### **1.4 Document Status and Schedule**

Version 1.0 is the first publication of the GUESS User's Guide. GUESS has been developed over a two year period from April 1, 1994-March 31, 1996. During these two years, GUESS was designed, encoded, and tested on a number of NASA and other scheduling applications. The current version of GUESS, running on the IBM PC or PC compatible Windows 3.1 or Windows '95 environment, was developed using Borland's C++ 4.5 and Microsoft's Visual C++ 1.5.

## **Guess User's Guide**

### **1.5 Documentation Organization**

This document is organized into 8 sections (including appendices). A short description of each of the sections follows:

*Section 1* Identifies the document and states its purpose and status.

*Section 2* Identifies related documents.

*Section 3* Provides an overview of the purpose and functions of GUESS.

*Section 4* Documents the installation procedures and initialization process of the software system for the new user.

*Section 5* Presents the software startup and termination procedures.

*Section 6* Describes each function with its corresponding operation.

*Section 7.* Contains a list of abbreviations and acronyms used in this guide.

*Section 8* Contains appendices related to this document.

## **2.0 Related Documents**

### **2.1 Parent Documents**

None

### **2.2 Applicable Documents**

1. NASA SBIR Phase I Final Report, Methodology and Mapping Between Problem Solving Requirements and Solution Scheduling Approaches in Mission Planning Expert Scheduling Systems, June 1993, AMEC/ Jay Liebowitz, Kensington, Md.
2. Scheduling, Objectives, Requirements, Resources, Constraints and Processes: Implications for a Generic Expert Scheduling System Architecture and Toolkit, June 1994, AMEC/Jay Liebowitz, Kensington, Md.
3. Conceptual Design of a Generic Expert Scheduling System Architecture and Toolkit, August 1994, AMEC/Jay Liebowitz, Kensington, Md.

## **Guess User's Guide**

### **2.3 Information Documents**

1. Looking Ahead Toward Testing of GUESS (Generically Used Expert Scheduling System), 1995, white paper, AMEC/Alisa Liebowitz, Jay Liebowitz, Chapman Houston, Vijaya Krishnamurthy, Kensington, Md.
2. "Intelligent Scheduling: Issues, Trends and Research Directions", white paper (based on an invited talk at the Goddard AI Conference, May 1995), AMEC/Jay Liebowitz, Kensington, Md.
3. "GUESS: Generically Used Expert Scheduling System", Proceedings of the Third World Congress on Expert Systems, Cognizant Communication Corporation /ISIS, New York, Feb. 1996, AMEC/ Jay Liebowitz, Alisa Liebowitz, Vijaya Krishnamurthy, Chapman Houston, Janet Zeide.
4. "Market Research and Program Report Requirements for Building a Generic Expert Scheduling System," March 10, 1995, prepared by James Martin Strategies North America Inc., prepared for American Minority Engineering Corporation.

## **3.0 Overview**

### **3.1 Background**

Scheduling is a prevalent function that is omnipresent throughout many industries and applications [1-4]. A great need exists for developing scheduling toolkits that can be generically applied to a number of different scheduling problems. To meet this need, more research is warranted for developing a state-of-the-art generic constraint problem-solver as related to scheduling.

Scheduling involves accomplishing a number of things that tie up various resources for a period of time. A scheduling problem can be defined as a set of constraints to satisfy. A solution to the scheduling problem is a set of compatible scheduling decisions that guarantee the satisfaction of the constraints [5]. Guaranteeing the compatibility of the decisions made is the role of constraint propagation. The order in which decisions must be made needs to be determined. In the NASA environment, scheduling is a critical area. According to the Engineering Services Group of McDonnell Douglas, in the next decade scheduling will be required for 365 days of the year and could take 2,000 to 3,000 people working continuously.

NASA has recognized a need for developing a generic scheduling toolkit. Toward this goal, NASA has developed such scheduling toolkits as PARR [6], AMP [7], Plan-It [8], and others [9]. Scheduling is a critical function for NASA Shuttle flights, payloads, and

## **Guess User's Guide**

crew members or scheduling scientists to use the NASA-supported satellites. The development of a generic expert scheduling system could ultimately be applied to many NASA and other scheduling applications. To better meet this goal, the American Minority Engineering Corporation (AMEC), has through NASA support, developed a generic expert scheduling system architecture and toolkit known as GUESS (Generically Used Expert Scheduling System).

GUESS has been designed to take advantage of an object-oriented, hierarchical architecture. GUESS contains two major levels of schedulers. The low-level schedulers are composed of different scheduling methods, mainly heuristic-based and optimization/algorithmic-based. The high-level scheduler, called the metascheduler, coordinates the activation of the low-level schedulers and injects any new information that is pertinent to the scheduling problem.

GUESS is designed to aid the human scheduler and to keep him/her in the loop. GUESS is a decision support aid as opposed to an automated replacement for the human scheduler. GUESS is programmed in C++ and runs on an IBM PC Windows environment.

An object-oriented approach has been used for GUESS in order to maximize the reusability and corresponding generality of GUESS. As an example, GUESS can schedule 2,551 events and over 14,000 constraints in under 45 seconds on a Dell 486 computer.

### **3.2 Major Features of GUESS**

#### **3.2.1 Suggestion Tabulator (Sugtab)**

The input file is first read. Depending on the technique chosen, the scheduling of the events takes place and the overall schedule is generated as output. Events are scheduled one at a time starting with an event of highest priority. To start, an event is scheduled by asking all of its constraints for their suggestions. A suggestion tabulator is given to an event, and the event passes it to each of the event's constraints. Each constraint can make zero or more suggestions to the suggestion tabulator, after which it can deduce the best beginning and ending times for the event based on the accumulated suggestions. The sugtab for an event is of four types: beginning range, ending range, beginning and ending equal suggestions. The range is controlled by greater and less than suggestions. Some constraints suggest a specific time for the beginning or end. An equal tabulator tabulates the equal condition suggestions and returns the value. The range tabulator tabulates the range by keeping a low and high limit.



## Guess User's Guide

### 3.2.2 Hill Climbing

This is another technique available for scheduling events in GUESS. The initial beginning time and satisfaction of the event is assumed to be the best and a non-linear search for better values is done on both sides of the initial value of time to search for better time and satisfaction. As the search transverses through either sides of the hill, the exponential increment for the time change can be adjusted for speed improvement. Similar to other techniques, the satisfaction of a particular event is calculated based on the satisfaction level of all its corresponding constraints. The higher the satisfaction level of an event, the happier it is in the schedule.

### 3.2.3 Genetic Algorithm

Genetic algorithms attempt to mimic the action of the “natural selection” process within living organisms to improve the characteristics of the species and to allow them to become better adapted to their environment. Genetic algorithms attempt to mimic this process within computationally difficult problems to arrive at some near optimum solution of the problem at hand. Genetic algorithms start with an initial population composed of some mixture of solutions that form the initial basis from which to begin a search for the best solution. This starter set of solutions is generally obtained in some arbitrary manner.

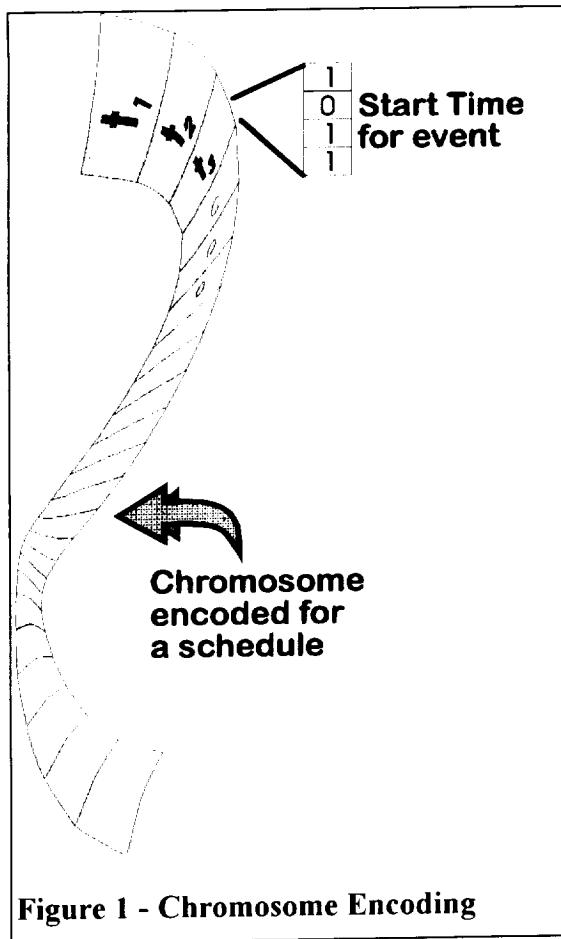
The algorithm then cross breeds this initial set of solutions combining the genes in the parent chromosomes to produce a child inheriting some characteristics from both parents. Random mutations are introduced during the breeding process to prevent the population from converging on some local maximum prematurely. Ranking of the solutions (chromosomes) is done using a fitness function which returns a positive number reflecting the relative value of the solutions. The higher the fitness number, the better the solution is. At each generation, the least fit solutions are removed from the gene pool. Eventually, we are left with a set of good solutions, from which the best solution is selected.

All of the specific domain knowledge required to implement the genetic algorithm is contained within the fitness function. This is an advantage over other AI techniques which require a large body of domain specific knowledge to be constructed. This information is time consuming to collect and limits the generality of the method.

#### *Guess Implementation*

Implementation of a genetic algorithm approach for *Guess* is done using the Evolutionary Object System (EOS) developed by Man Machine Interfaces, Inc. EOS is a C++ class library for creating genetic algorithms. The first decision that must be made is how to encode the schedule information as a chromosome. The standard encoding used in most classical genetic algorithm work is the binary encoding. Each gene within the chromosome is a series of bits. The genes are linked together to build a long chromosome (See Figure 1).

## Guess User's Guide



**Figure 1 - Chromosome Encoding**

EOS supports a variety of other possible encodings; however, the binary encoding is the most generic and was also recommended by the EOS vendor; so the decision was made to use the binary encoding in the implementation of a genetic algorithm within *Guess*. The only significant variable information associated with the schedule is the time at which each event is scheduled to occur. *Guess* assumes that the duration of an event is fixed and determined by the user at the time the event is entered; therefore, the only significant information that must be associated with the event is the starting time of the event. Each event has a specific gene within the chromosome. The gene occupies a series of bits of size *tTime* which is sufficient to record the starting time of an event.

The standard seeder supplied with EOS creates the initial population in a random manner. This approach does not take advantage of any re-ordering that may have

occurred previously and always starts from a clean slate. A new class (*TGuessSeeder*) was derived from the *TRandomSeeder* class. This class makes the first member of the population the original starting schedule and then randomly populates the remainder of the chromosomes.

A special class, *TSchedPheno* was derived from the abstract class *TPhenotype* to provide translation between the genotype (chromosome) and the expression of the genotype or phenotype. In the case of *Guess*, the expression of the genotype is the starting times of each event within the schedule.

*TguessGA* was derived from the *TBasicGA* class to allow some customization and statistics reporting on the genetic algorithm. This class maintains a copy of the best individual obtained so far as well as keeping track of the satisfaction scores and number of unscheduled events in each generation.

The calculation of fitness comprises the core of the genetic algorithm. The basis of calculating the fitness was chosen to be the satisfaction factor. A member variable that points to a genotype was added to the *cSchedule* class. The other basic change was in the calculation of the event times. Normally in *Guess*, the event times are extracted from

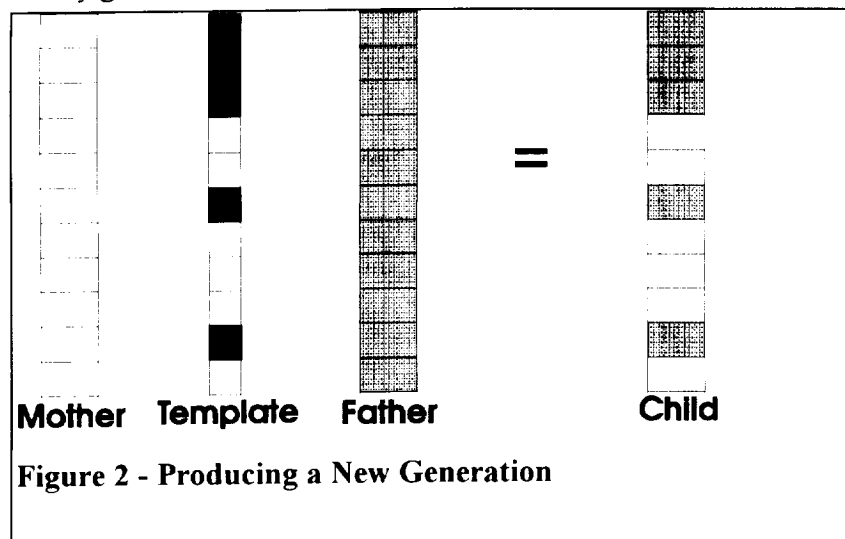
## Guess User's Guide

internal member variables within the event object. This is not suited for processing the genetic algorithm, as the genetic algorithm requires evaluating the fitness of a large number of competing schedules for each generation. The `mBeg` member function was therefore modified to check the `cSchedule` and if a genotype was provided, the event starting time was extracted from the genotype rather than using its internal value. If no genotype is provided, then the event times are calculated in the usual manner.

The `mEnd` and `mDuration` member functions were similarly modified to retrieve their times based upon the beginning time obtained from `mBeg`. This proved to be a rather elegant and efficient method of adapting the original *Guess* algorithms to work with a genetic algorithm with a minimal amount of change and without breaking any of the previously debugged and tested code.

The fitness function is based on using the schedule satisfaction normalized to a positive number. A division by the number of events that cannot be scheduled (due to resource conflicts) is done to impose a stiff penalty to resource constraint violations. Without the penalty, the genetic algorithm tends to produce solutions with unschedulable events with alarming frequency.

An additional menu item was added to the Schedule menu to allow selection of either the suggestion tabulator or genetic algorithm as the solution method by checking the appropriate item in the menu. To aid in the debug and optimization of the algorithm, a few additional dialog boxes and menu items were added. First, a dialog was added to allow the genetic algorithm parameters to be changed interactively. This was convenient as it was anticipated that a large number of scenarios would be run while varying the basic genetic algorithm parameters to try and find an optimum combination. Additionally, a debug dialog was added allowing the user to view the results of the genetic algorithm on a generation by generation basis.

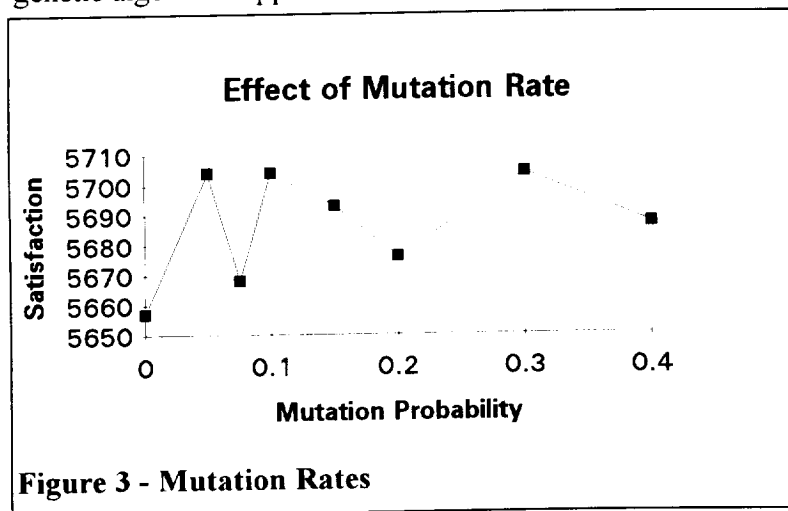


## Guess User's Guide

After a little experimentation, it appeared that the best approach appeared to use an elitist replacement strategy with a uniform crossover breeding approach. The elitist replacement strategy replaces the worst individuals in the next generation with the best individuals from the current generation. This strategy was used because it appeared during preliminary analysis that some of the good solutions were being lost in the reproduction process.

Uniform crossover mates two chromosomes by introducing a randomly generated crossover mask. Genes represented by bits set in the mask are taken from the mother; genes represented by cleared bits in the mask are taken from the father. This appeared to give the best results in the trials.

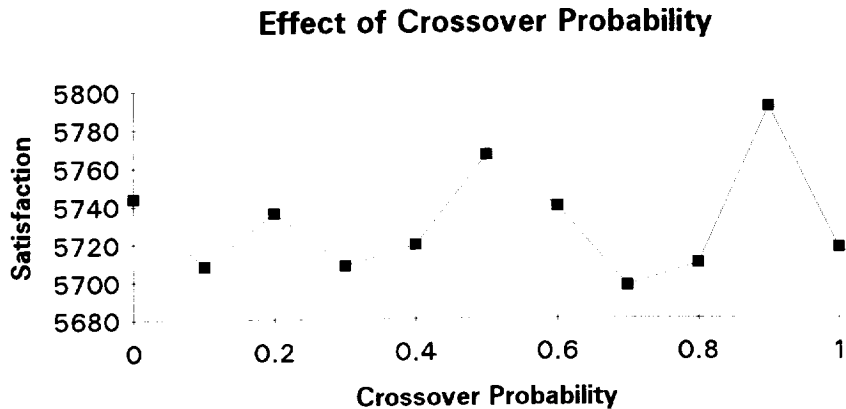
One important thing to remember is that genetic algorithms by their very nature are random and do not tend to yield reproducible results. This was borne out in the testing. Whereas, given the same input, the suggestion tabulator always arrived at the same answer; the genetic algorithm usually yielded widely varying answers even though the satisfaction values may have been similar. Various studies were done on a 50 event schedule to determine the effects of varying the mutation rate and crossover probabilities. These results are reflected in Figures 2,3 and 4. Neither one of the graphs shows any clear cut trend or optimum especially in the case of mutation rate. It appears, however, that the genetic algorithm approach is useful for 100 events or less.



The best values appear to be 0.05 for the mutation rate and 0.9 for the crossover probability although there appears to be such a wide variation in the numbers that the use of the values as optimal appears

questionable. The genetic algorithm yields very tight compact schedules with events scheduled in the minimum time span required. By contrast, while satisfying the schedule constraints, the traditional methods, namely suggestion tabulator and hill climbing tend to do so by spreading out the schedule and increasing the schedule makespan.

## Guess User's Guide



**Figure 4 - Crossover Probabilities**

The length of time that the genetic algorithm takes is due mainly to the need to calculate schedule satisfaction for multiple schedules each generation. Compounding this is the requirement to process many generations before arriving at the solution. By contrast, the hill climbing and suggestion tabulator deal with the schedule in a much more localized manner. Each event has its satisfaction calculated individually and is then moved accordingly resulting in fewer satisfaction calculations in arriving at a final solution.

### 3.2.4 User Interface Design

The displays must be familiar and easily recognized by the users. They must also allow the user to grasp any change in the data and the significance of that change instantly.

To aid in user comprehension, GUESS is implemented in the broadest terms of flexibility. Its unique design allows the user to alter the organization of the GUESS display and desktop. By altering the look of GUESS to one that is more familiar to the operators, one can save much of the time they would spend adapting to the new system. This would also significantly reduce the learning curve for new operators.

To that end, we have put considerable resources behind the development of Graphical User Interfaces, such as Visual Basic Xtensions (VBX). These systems allow us to display information in simple two or three-dimensional charts of many different types. This will allow the user to “customize” the interface to their own personal tastes. In a mission-critical atmosphere, the user will be able to select the charts that he or she can best understand quickly, making the interface far more useful. GUESS will remember those changes and present the interface in that fashion until told otherwise. So once the system is “customized”, there is no need to repeat the effort. For a system with multiple users, multiple environments can be set so that each user would have his/her own individual “look” if he/she wished.

## Guess User's Guide

GUESS avoids the complexity inherent in many scheduling systems by letting the user decide how much of the system he wishes to use. Instead of having to wade through a thick book of directions, GUESS is designed in an intuitive fashion and conforms to the GUI standards laid out by the industry.

GUESS is designed with a stair-step approach to learning. From the start, the user can run the system on pre-set defaults, if that is what he wants; there is no need to change anything.

This is the first level of GUESS operation. When he wishes to add other options, the functions in GUESS can be added one at a time so a new user doesn't get overloaded with all the changes. They can also be added in groups for power-users who are already familiar with the system. This puts the complexity of the system in the hands of the user.

The principal idea behind the GUESS system was to build a system so flexible that it would fill a broad spectrum of needs. This flexibility is implemented in the "Loosely Bound" architecture that comprises the GUESS system. Instead of having one rigid program, GUESS relies on a series of programs related by a message loop.

GUESS capitalizes on this open-architecture by allowing for the addition of other programs to be added to the message-loop. New programs or additional modules can be added to the system simply by loading them to the GUESS directory. The main GUESS program will scan the local directory and link all the modules found. If a specific module is needed, it will give a warning about the omission.

By using the Object-Oriented Approach, GUESS allows the user to become an active part of the system and alter major parts of the interface. This segmenting of the program code makes for a more compact program with less extraneous code. This eliminates a lot of operational overhead and helps the memory burden on the computer. Only the sections that are needed have to be loaded into the system.

### 3.2.5 Resource Modeling

Almost all scheduling problems depend in part on the availability of the necessary resources being present at the time of execution of a scheduled event. Some problems are primarily resource constrained while in other problems there are a sufficient abundance of resources such that resources do not seriously constrain the scheduling. For GUESS, it was deemed necessary to add a generic resource model, capable of handling most resources that could conceivably be used in scheduling situations.

## Guess User's Guide

For purposes of scheduling there appear to be two generic types of resources that could be used to model most resources used within a scheduling scenario. These are termed "Binary" and "Depletable" resources respectively. A binary resource is a resource of which a fixed amount is available at any given time, the resource is not depleted or consumed by the events utilizing it. An example of a binary resource might be the crew members on a space shuttle mission. Crew members may be utilized on a specific task but are not depleted or destroyed by the event. They are immediately available to tackle another task on completion of the present task. The function of the scheduling algorithm is to make sure that these binary resources are not overcommitted at any given point in time.

Depletable resources, on the other hand, are in fact consumed by tasks using them. As different events occur, these events utilize a portion of the resource until the resource is completely consumed and there is nothing left to consume. As well as consuming the resource, it is possible for certain events to replenish the resource. Typical systems that might be modeled by this approach include the consumption of propellant by a spacecraft as a result of various maneuvering events or the drain on the spacecraft battery caused by the operation of electrical equipment. In the case of the propellant, it is unlikely that events would occur to replenish the resource--in the case of the battery, recharging could occur by orienting the solar panels toward the sun.

The function of the resource scheduler is to schedule the activities to prevent depletion of the resource before all of the required activities are complete. In the case of resources that may be recharged, the scheduling algorithm must schedule sufficient recharging events between uses to keep adequate resources in hand to perform the necessary consuming activities. This involves alternating recharging and consuming activities in some pattern to sustain the resource.

These two types of resources are the most generic resources that can be used within a scheduling context; however, it was also felt necessary to allow the addition of custom resource models, that could be user-designed yet easily integrated within the GUESS scheduling engine. These custom models could be integrated as Dynamic Link Libraries (DLL's) and attached on the fly to the GUESS executable. This would allow unlimited flexibility for GUESS and would relieve the designers of having to anticipate unusual resource models or bloating the code in trying to incorporate every conceivable resource model within GUESS.

This choice of models provided a suitable methodology for modeling of resources within GUESS while providing plenty of room for future expansion. The next decision was the scheduling algorithms to be used in satisfying the resource constraints and their relationship to the algorithms already being used to schedule other constraints within the schedule. Rather than attempting to build an external resource model, it was felt to be beneficial to integrate the resource constraint satisfaction mechanism within the existing constraint satisfaction methods.

## Guess User's Guide

There are two reasons for deciding upon this approach:

- Resource constraints are in fact just another form of constraint and it is beneficial to treat them in the same manner as other constraints from both a computational efficiency stand point as well as eliminating the need for doing several iterative calculations going from resources to constraints, in a back and forth motion, until the optimal solution to both problems is found.
- Having a single algorithmic model makes for a very elegant solution. In fact the same optimization techniques can be used--the difference between the two types of constraints lies in the computation of its satisfaction score. The optimization algorithm need not be different. The "satisfaction criteria" is the only difference between the two.

The main emphasis in implementing a resource model for GUESS becomes a matter of determining the appropriate functions to use in computing resource satisfaction. The resource satisfaction should be a low value indicating a lack of satisfaction and forcing the offending event to be rescheduled if the event makes the resource usage exceed the total allowed. The resource usage if below the maximum allowed is an acceptable state of affairs and should give a satisfaction rating consistent with this. For a binary resource, maximum resource usage should be encouraged (as long as the maximum limit is not exceeded), since this will tend to decrease the schedule span and make use of the resource.

For a depletable resource, the situation is not quite so clear cut, since once a resource is gone, it must be either recharged (if it can be) or the resource is unusable for future activities. Therefore, the same positive satisfaction value is returned for all depletable resources whose usage is below the limit. A further simplifying assumption was made that resource usage is constant throughout the duration of the event. To get a profile of resource usage, resource usage is sampled throughout the duration of each event. A resource constraint satisfaction is obtained for each resource utilized by an event. A binary resource constraint will compute usage by integrating the resource usage over the time period of the event.

The algorithm first discards all events which do not overlap with the time period of interest. Then an adjustment is performed to obtain an adjusted average over the period of time for which the event overlaps. The implicit assumption is made that resource usage is linear over the event duration and multiple parallel events affect depletion in a linear additive fashion.

Usage for a depletable resource is calculated in a similar manner as that for a binary resource with the principal difference being that all events scheduled prior to the beginning



## **Guess User's Guide**

of the time period of interest must also be summed . In this instance, an overlapping event is defined as an event whose start time is less than the end of the interval of interest.

To allow the resource model in GUESS to be readily extensible a feature for allowing the addition of custom resources was incorporated. The most flexible approach to doing this is to allow the user to add a custom dynamic link library which can be linked on the fly to the GUESS application. Doing so requires a standard application programming interface (API) to be developed to allow communication between the DLL and the GUESS engine. This requires a definition of entry points and types to be utilized within the DLL. The GUESS executable loads the DLL using the standard Windows function LoadLibrary function and then attempts to load each of the exported entry points by name. GUESS loads each of the entry points by name rather than the number even though loading by number is slightly more efficient--loading by name is easier to use and manage. As GUESS looks up the function pointers when the DLL is initially loaded, this causes an imperceptible performance penalty. If GUESS either cannot load the library or cannot find one of the required functions, then an error message is reported.

When a resource is over committed and needs to be rescheduled, the suggestion preferred is to reschedule the current event to occur just after the last event using the resource. Since events are processed in priority order, this forces lower priority items to be scheduled later than high priority events. A more sophisticated means of producing suggestions could be included as a later enhancement; however, for the test cases considered, the suggestion strategy yielded acceptable results.

### **3.3 Performance of GUESS**

GUESS has been tested in a number of applications. For NASA applications, such as scheduling satellite experimenter requests to use the NASA supported satellites, GUESS can schedule, for example, 2,551 events and over 14,000 constraints in 45 seconds on a Dell 486 computer. This performance corresponds well with other NASA expert scheduling systems that can schedule up to 6,000 events in 2.5-3 minutes.

### **3.4 Future Direction of GUESS**

#### **3.4.1 Developing Database Interfaces to GUESS**

One of the major future efforts for GUESS is to develop database interfaces to GUESS for ease of inputting the events, constraints, and resources. Currently, the user has to enter the events and associated resources and constraints into GUESS one-by-one. For large scheduling cases (especially in the NASA environment of several thousand

## **Guess User's Guide**

events), this could be a time-intensive effort. For commercialization of GUESS, there is a need for establishing database links with popular database packages (e.g., Access, etc.) (and even spreadsheet packages like Excel) for an efficient way of entering the data for scheduling. We currently have a test case generator to facilitate the generation of multiple events, constraints, and resources. However, for customization purposes, establishing the database/spreadsheet links are essential for enhancing GUESS during the Phase 3 commercialization effort.

### **3.4.2 Categorize Different Scheduling Problems and Develop Generic Scheduling Models for Each Within GUESS**

In testing the current version of GUESS, various types of scheduling problems were used. These included: an Army strategic scheduling problem of assigning units in a deployed theater; scheduling Army battalion training exercises; scheduling City of Rockville Pee Wee and Midget baseball games; scheduling Department of Computer Science courses and corresponding sections for Montgomery College; and scheduling NASA experimenter requests to use NASA-supported satellites. We have included some scheduling frameworks within GUESS to facilitate different types of scheduling problems (e.g., timetabling, game scheduling, classroom scheduling, job shop scheduling, etc.). However, we need to further develop these frameworks and include within GUESS other frameworks based on the type of scheduling problem. In this manner, GUESS will include the framework and user interface for different types of scheduling problems. This should also allow improved efficiency in inputting the scheduling data for a particular class of scheduling problems.

### **3.4.3 Develop a Test Suite of Different Types of Scheduling Cases to Run Against GUESS Each Time a New Scheduling Technique is Included in GUESS**

We currently have a useful test generator that helps in developing test cases for GUESS. However, we need to have a more complete test suite of different types of scheduling cases to run against GUESS each time a new scheduling technique is included in GUESS. This test suite should include different classes of scheduling problems that are heterogeneous in order to test the new scheduling methods within GUESS for effectiveness and efficiency. Manufacturing scheduling test cases need to be added to our test suite.

## **Guess User's Guide**

### **3.4.4 Improved Methods for Customizing the Report Generation Function of GUESS**

Through the Gantt.VBX in GUESS, we have some control in customizing the scheduling report. We can perform sorting by event name, date, resource, etc., but we need to have other options available for customizing the scheduling report for the user. For example, the athletics scheduler may want to have the scheduling report by baseball field, team, player, week, game, etc. and GUESS should be flexible enough to handle these demands. Customizing the report generation function for the user will also help in user acceptance of GUESS.

### **3.5 Summary**

In the coming years, the major trends in intelligent scheduling systems are:

- the majority of the AI approaches to scheduling will continue to be constraint-based;
- movement will continue toward expert scheduling system shells/generic constraint-based satisfaction problem-solvers;
- interest will expand in object-oriented/agent-based programming paradigms and hierarchical architectures used in intelligent scheduling systems;
- increased use will occur of hybrid intelligent systems for scheduling (knowledge-based, neural networks, fuzzy logic, genetic algorithms, optimization/operations research, etc.)

We feel that GUESS is an effort that supplements well these future directions in intelligent scheduling. More testing for generic scheduling qualities of GUESS will be conducted in the near term, as well as expanding the number of scheduling techniques in the GUESS toolkit.

### **3.6 References**

1. Pinedo, M. (1995), *Scheduling Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, NJ.

## Guess User's Guide

2. Morton, T. and J. Pentico (1993), Heuristic Scheduling Systems, John Wiley, New York.
3. Zweben, M. And M. Fox (eds.) (1994), Intelligent Scheduling, AAAI/MIT Press, Cambridge, MA.
4. Brown, D. And W. Scherer (eds.) (1995), Intelligent Scheduling Systems, Kluwer Publishers, MA.
5. Noronha, S.J. and V.V.V. Sarma (1991), "Knowledge-Based Approaches for Scheduling Problems: A Survey, "IEEE Transactions on Knowledge and Data Engineering, IEEE, Vol. 3, No.2, June.
6. NASA Goddard Space Flight Center (1994), Proceedings of the 1994 Goddard Conference on Space Applications of Artificial Intelligence, Greenbelt, Maryland, May.
7. Stolte, A. (1994), An Object-Oriented Approach to Scheduling, AI Expert, Miller Freeman Publications, San Francisco, CA.
8. Lee, J.K., M. Fox, and P. Watkins (1993), Special issue on "Scheduling Expert Systems and Their Performances, "Expert Systems With Applications: An International Journal (J. Liebowitz, Ed.), Elsevier/Pergamon Press, New York, Vol. 6, No.3.
9. Liebowitz, J., P. Lightfoot, and P. Dent (1991), " Conflict Resolution Strategies in Expert Scheduling Systems: Survey and Case Study, "The Knowledge Engineering Review, Cambridge University Press, England, Vol. 6, No.4.

### 4.0 Installation

Guess consists of a main program file guess.exe and three dynamic link libraries and one Visual Basic extension (VBX) file. Guess is designed to run under the Microsoft Windows 3.1 operating system. The program file guess.exe should be installed in the program directory (typically c:\guess although the user may choose to install in any directory that is convenient) and the DLL and VBX files should be installed in the windows system directory (typically c:\windows\system but may vary if a custom installation of MS Windows has been done). The files are as follows:

*Program Directory (i.e. c:\guess)*

guess.exe

*Windows System Directory (i.e. c:\windows\system)*

ctl3dv2.dll

eos20.dll

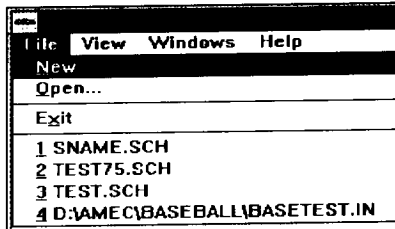
ganttvbx.vbx

tabstcl.dll

## Guess User's Guide

A program group and icon should be created for Guess. Refer to the MS Windows user manual for details on creating program groups and icons within the Program Manager.

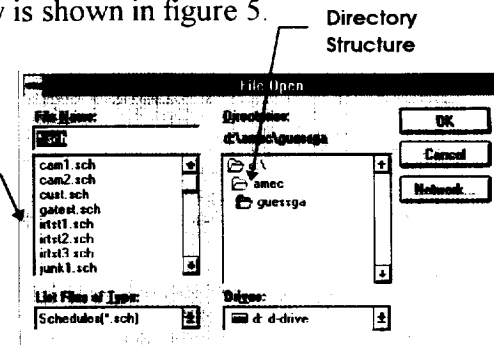
### 5.0 Working with Guess



**Figure 5 - Main Window Menu**

alternative schedules and scheduling algorithms that can be used within Guess. The menu that appears initially in the main window is shown in figure 5.

The user may choose New to get a blank schedule on which to begin entry of data for a new schedule or choose to open an existing schedule. The four most recently accessed schedules appear on the bottom of the menu for easy selection. The user may choose to open any arbitrary schedule using the Open... menu selection bringing up the file dialog (see figure 6). A file may then be selected using the drive and directory structure boxes to navigate and locate the desired scheduling file. Guess schedule files have the extension .sch by default although the user may override this selection by choosing "All Files" in the "List Files of Type" drop down list box.



**Figure 6 - File Open Dialog**

The network button appears if Windows is being run in a network environment and allows the user to connect network drives to access Guess schedule files stored remotely. Clicking on the OK button opens the file for editing.

The network button appears if Windows is being run in a network environment and allows the user to connect network drives to access Guess schedule files stored remotely. Clicking on the OK button opens the file for editing.

### 6.0 Using the Guess Scheduling Program

#### 6.1 Viewing the Schedule

The schedule appears in a three paned window as shown in figure 7. In the upper left hand corner appears a tabular listing of the events in the schedule. In the upper right hand

## Guess User's Guide

section the schedule appears in gantt chart form. In the bottom of the window the resources appear as a graph of resource usage over time.

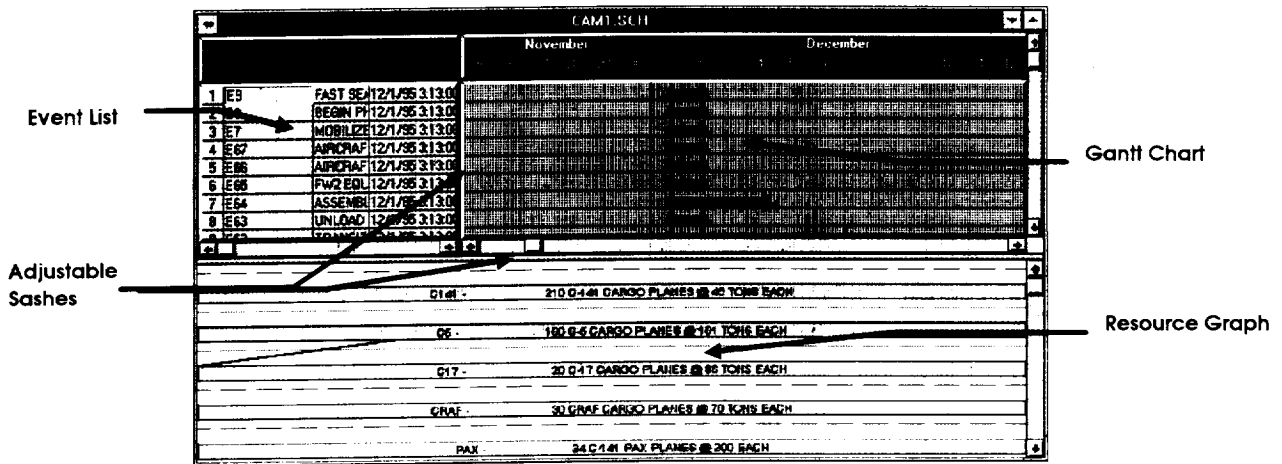


Figure 7 - Schedule Window

Each of the three panes is easily adjustable by dragging the sashes located on the pane boundaries to the desired location. This allows any of the window panes to be adjusted to fit viewing preferences.

The individual panes are also scrollable allowing viewing of any portion of the display even when the whole display is too large to fit within the window pane.

### 6.2 Event List Pane

The event list panes gives a tabular listing of the events (see figure 8). Shown in the event pane is the event name, description, start date and time, end date and time, duration and satisfaction. Satisfaction measures how well the event as scheduled meets the constraints placed upon it. The satisfaction value ranges from -127 to +127 with a +127 meaning that all of the constraints are perfectly satisfied.

## Guess User's Guide

#	Event	Description	Start	End	Duration	Satisfaction
1	E9	FAST SEALIFT SHIPS AVAILABLE	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	-20
2	E8	BEGIN PHASE II OF CRAF	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	-15
3	E7	MOBILIZE CRAF PHASE I	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	42
4	E67	AIRCRAFT AVAILABLE, N+9	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	3
5	E66	AIRCRAFT RETURN TRANSIT TO US, 1 DAY	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	127
6	E65	FW2 EQUIPMENT AND PERSONNEL IN PLACE @ N+11	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	-56
7	E64	ASSEMBLE UNIT, 3 DAYS	12/1/95 3:13:00 PM	12/4/95 3:13:00 PM	3 Days 0 Hours	-56
8	E63	UNLOAD FW2, 1 DAY, N+8	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	127
9	E62	TRANSIT TIME TO KOREA, 1 DAY	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	127
10	E61	LOAD FIGHTER WING #2, 1 DAY, WEST COAST, N+6	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	-63
11	E60	WAIT FOR AIRCRAFT AVAILABILITY, 4 DAYS	12/1/95 3:13:00 PM	12/5/95 3:13:00 PM	4 Days 0 Hours	-59
12	E6	MOBILIZE SEALIFT	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	127
13	E56	AIRCRAFT AVAILABLE, N+5	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	43
14	E55	AIRCRAFT RETURN TRANSIT TO US, 1 DAY	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	127
15	E54	BW2'S EQUIPMENT AND PERSONNEL IN PLACE @ N+7	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	-56
16	E53	ASSEMBLE UNIT, 3 DAYS	12/1/95 3:13:00 PM	12/4/95 3:13:00 PM	3 Days 0 Hours	-56
17	E52	UNLOAD BW2, 1 DAY, N+4	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	127
18	E51	TRANSIT TIME TO KOREA, 1 DAY	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	127
19	E50	LOAD BOMBER WING #2, 1 DAY, EAST COAST, N+2	12/1/95 3:13:00 PM	12/2/95 3:13:00 PM	1 Days 0 Hours	11

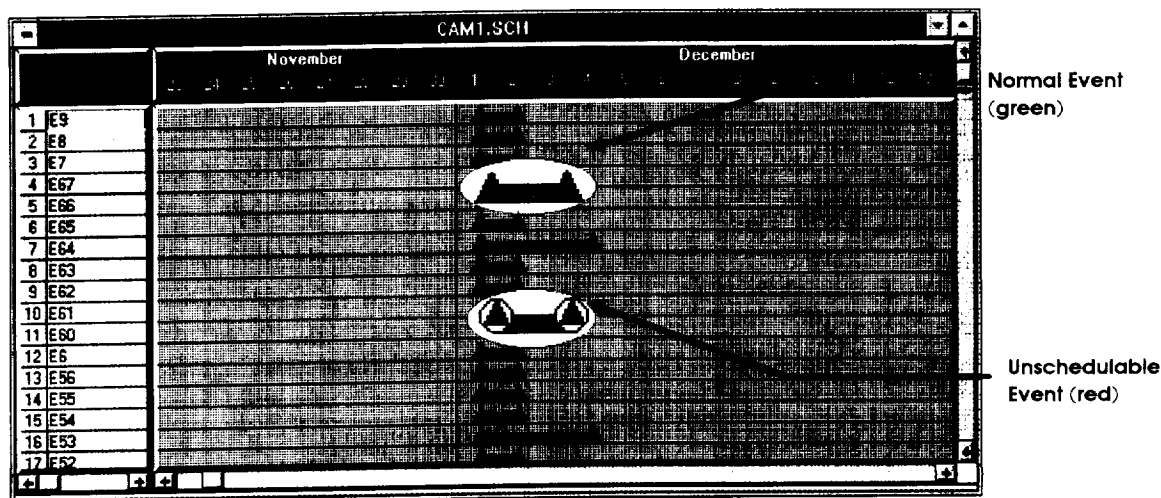
**Figure 8 - List Pane**

An event may be edited by double clicking on it. This brings up the event dialog described in "editing events".

### 6.3 Gantt Chart Pane

The Gantt Chart pane displays all of the events in a traditional gantt chart view (see figure 9). Starting times and ending times are shown on the timescale. The timescale for the gantt chart may be adjusted using the View menu.

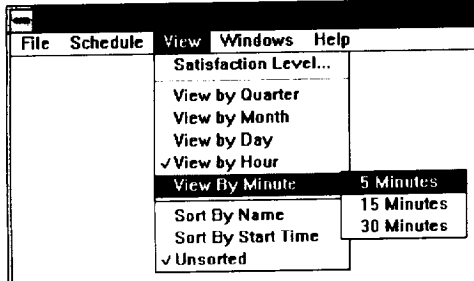
Events may be edited by clicking on the gantt chart bar. The event times may be modified by dragging the bar with the mouse. The event duration may be changed by dragging on either the bar starting or ending symbol with the mouse.



**Figure 9 - Gantt Chart Pane**

## Guess User's Guide

Events meeting the scheduling constraints are shown in green; events not meeting the satisfaction criteria are shown in red and have a different starting and ending symbol than do events that satisfy the scheduling constraints. Events that have resource scheduling conflicts are always shown in red; the user can specify minimum satisfaction levels below which events are flagged as unschedulable using the Satisfaction Level menu item.



**Figure 10 - View Menu**

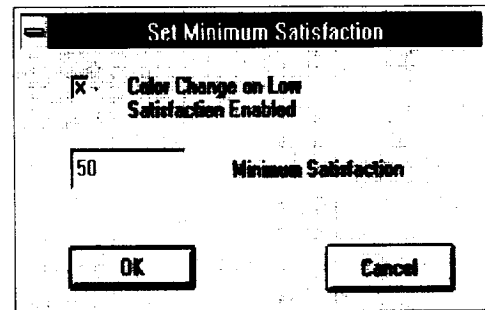
The gantt chart view can be customized using the view menu. Selecting "Satisfaction Level..." produces a dialog.

This dialog allows the minimum satisfaction level below which events are flagged in red to be set. Checking the "Color Change on Low Satisfaction Enabled" causes the color of the bar to change to red when the event satisfaction drops below this level. If this box is not checked then the bar color

changes to red only when the event has resource constraints which are not satisfied. The minimum satisfaction must be in the range of -127 to 127.

The user is given the option to change the timescale as appropriate. The timescale may be displayed as quarters, months, days, hours and minutes. The minutes option has a further choice of interval defined by a sub-menu.

Events may also be sorted either alphabetically or by order of their start date by checking the appropriate menu item on the View menu. The gantt chart will be automatically resorted after an item is checked. Selecting unsorted will cause the gantt chart to resort to its original unsorted order.



**Figure 11 - Satisfaction Dialog**

### 6.4 Adding/Editing Events

When starting a new schedule a World event should be first created. This allows specifying the duration of the schedule and constraining all of the events to occur within the specified duration. This event should be created as a locked event to prevent it from being shifted in time with no constraints or resources attached to it.



## Guess User's Guide

Events may be added to the schedule using the New Event menu item under the Schedule menu. Existing events may be edited by double clicking anywhere along the event's row in the gantt chart or event list. Both actions bring up the event dialog as shown in figure 12.

Along the top of the dialog is a set of tabs. These allow access to all of the event constraints. A unique name without blanks must be entered for the event name. This should be something meaningful to the user as this is the primary identification used for the event. The priority of the event may be set from 1 to 10. Ten is the highest priority and one is the lowest. The scheduling engine will attempt to resolve the constraints of high priority events first before considering lower priority events.

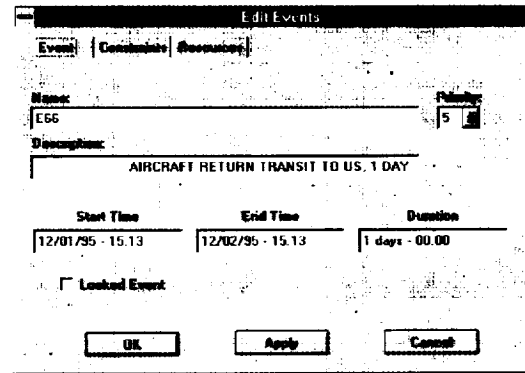


Figure 12 - Event Dialog

Pressing the OK button incorporates the event and all modifications made to it into the schedule, removes the dialog box, and updates the schedule views. Pressing Apply includes the modifications into the schedule and updates the displays but does not remove the dialog box making it available for further editing. Pressing Cancel removes the dialog and nullifies the effect on any edits made. If "New Event" has been chosen then the event is not added to the schedule.

### 6.4.1 Editing the Event

A short description of the event may also be entered if desired. Start time, end time and duration can also be entered for the event. Start and end times are entered in the format mm/dd/yy HH.MM where mm is the month, dd is the day of the month, yy is either the two or four digit year, HH is the hour (on a 24 hr clock) and MM is the minute. Duration is entered in the format dd days - HH.MM. As these variables are entered Guess automatically recalculates the remaining fields as necessary.

A *locked* event is an event that is fixed in time and cannot be rescheduled by Guess. This class of event is used for milestone events whose time is fixed and is used as a constraint for other events. Checking the locked event box makes this a locked event.

## Guess User's Guide

Figure 13 - Editing Constraints

Using the tab control at the top of the dialog the user can readily switch between editing the event itself, the event's constraints and resource usage. If this is the only event in the schedule then the constraint tab will be disabled as there exist no events with which to constrain the current event. Similarly if no resources exist, then the resource tab will be disabled.

### 6.4.2 Editing the Constraints

Clicking on the constraint tab changes the dialog display to show the constraints associated with the event. If there are no constraints associated with the event then all of the controls with the exception of the "new" button will be disabled. To add a new constraint click on the "new" button and a blank constraint form with all of the applicable controls enabled.

Checking the delete box will cause the constraint to be deleted after either OK or Apply is selected. For events with multiple constraints, the additional constraints may be located using the constraint navigation arrows located on the right side of the dialog box.

The constraining event is the event that provides the constraint to this event. It must be an event that already exists in the schedule. The weight can be adjusted to reflect the importance of the constraint. When scheduling, Guess uses these weights to give more emphasis towards satisfying constraints having higher weights than those having lower weights. A short description of the constraint can be entered in the description box.

The type of constraint must be entered in the "Type" dropdown list on the left side of the dialog. The types of constraints supported by Guess are as follows:

Constraint Types	
Type	Description
After	This event must occur after the constraining event.
Before	This event must occur before the constraining event
Begin With	This event must begin at the same time that the constraining event begins.
During	This event must occur at some point during the constraining event.
End With	This event must end at the same time as the constraining event.
Meta	This allows more complex constraints to be built between two events. See text for explanation.

## Guess User's Guide

If a World event has been created (see section 6.4) , then a “during” constraint needs to be created for each event forcing the event to occur within the world event. Constraints establish the relationships between the events in the schedule.

As an example, let us try to schedule a vacation trip. The first step is to set up a world event to define the time period during which we must complete all of the activities associated with the trip. Let us say that we wish our trip and all activities associated with the trip to be completed within the month of June. The world event would then be entered into the schedule with a start date of June 1 and an end date of June 30. This event would be a locked event since its endpoints are fixed and it is not permissible for Guess to reschedule the event. This event would not have any constraints as it is not dependent upon any other event. Each subsequent event added to the schedule would be tied to this world event using a *during* constraint.

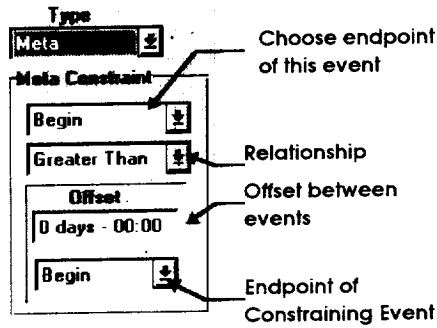
The first event might be to pack luggage. Therefore an event is created which we entitle `pack_luggage`. The next event would be to travel to the airport. We name this event `depart_to_airport`. Obviously, the luggage must be packed before departure to the airport. Therefore, we add a constraint to the “pack luggage” event to the effect that it must come before the “depart to airport” event. To do this, we double click on the “pack luggage” event in the gantt chart pane. Next we select the *Constraints* tab to go to the Constraint dialog for the event. Since there are no constraints as yet entered for this event all of the controls with the exception of the *New* button and the *OK*, *Apply* and *Cancel* buttons on the bottom are active.

To create a new constraint button, we click on the *New* button. This activates all of the controls on the form. Since this constraint must occur before `depart_to_airport` we select “Before” from the dropdown list associated with the type field in the upper left corner of the dialog. `depart_to_airport` is entered as the constraining event. The weight field allows for an entry of 1 to 100 with 1 being the least weight and 100 being the most weight. For this particular constraint we choose a midrange weight of 50.

Now we must add the “during” constraint for the world event. To do so, we again click on the *New* button to add another constraint. “During” is chosen as the constraint type and “World” is chosen as the constraining event (We can in reality name the world event anything we choose but for clarity we are calling it World.). We can now click *OK* to add the constraint and dismiss the dialog.

To make the constraints symmetric and to produce a tighter schedule, an “after” constraint should be associated with the `depart_to_airport` event. We follow the same procedure as before to add a constraint to the `depart_to_airport` event. We double click on the `depart_to_airport` event in the gantt chart and then select the *Constraint* tab. This time we select “after” as the constraint type and enter “`pack_luggage`” as the constraining event.

## Guess User's Guide



**Figure 14 - Entering a Meta Constraint**

Meta constraints allow more complex relationships to be developed between events. When meta is chosen as the constraint type the group of controls labeled “Meta Constraint” is made active. The endpoints used to establish the constraining relationship for both the constrained event (the event being edited) and the constraining event must be chosen.

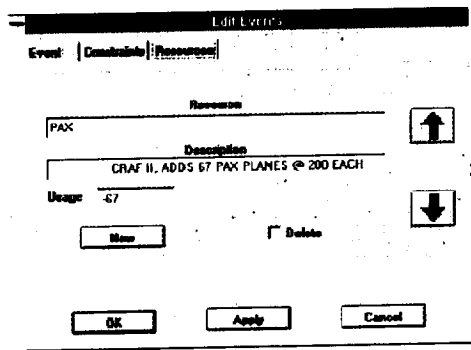
In addition, a relationship must be chosen. This can be either greater than, less than, or equal..

A time offset between the selected endpoints of the constrained event and the constraining event should also be chosen.

Using the trip scheduling example of the previous section, it is easy to demonstrate the use of a meta constraint. For example, suppose we needed a passport for our trip. Since it takes two weeks to obtain the document from the passport office, we must send the application in at least two weeks before we depart to the airport. Therefore, we can show this relationship on the schedule by adding a meta constraint to the `send_passport_application` event. The beginning of this event must occur two weeks before the beginning of the `depart_to_airport` event. Going to the *Constraint* tab of the `send_passport_application` event, we select meta constraint as the constraint type. This activates the meta constraint controls on the left hand side of the dialog. `Begin` is selected in the top box since the beginning of this event must occur two weeks before `depart_to_airport`.

The beginning of the `send_passport_application` must be less than the (before) the `depart_to_airport` event so less than is chosen as the relationship. Since the relationship is from the beginning of the constrained event (`send_passport_application`) to the beginning of the constraining event (`depart_to_airport`) we select “`Begin`” in the third box. The offset is 14 days (two weeks).

### 6.4.3 Editing the Resource Constraints



**Figure 15 - Editing Resource Constraints**

The next step is to edit the resource constraints. This is accessed by clicking on the resource tab in the event dialog (see figure 15). If there is more than one resource associated with the event, then the navigation arrows on the right of the dialog may be used to access the other resource constraints in the same way that they are used in the constraint tab.

The resource name needs to be entered on the first line. The resource must already exist in the schedule, otherwise, Guess will give an error message. You may enter a short description of the

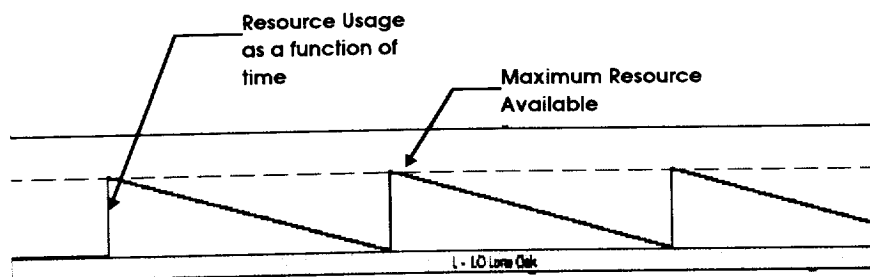
constraint and indicate the resource usage by the event.

If no resource constraints exist for an event the dialog will appear with all of the controls disabled except for the "new" button. A new resource constraint is created by clicking on the new button. This activates the controls and allows entry of a new resource constraint. The "new" button may be used at any time to add a new resource constraint.

Checking the delete box will cause the resource constraint to be deleted once either Apply or OK is selected.

### 6.5 Resources

The resource pane is shown as a set of five resource graphs showing resource usage over time. The time interval is the schedule makespan, i.e. the time period between the start of the earliest event in the schedule to the time at which the last event is completed.



**Figure 16 - Resource Graph**

The graph shows the resource usage as a percentage of the maximum available resource. The maximum available resource is shown as a dotted red line. Five graphs are shown in the resource pane. The pane may be scrolled to see all of the resources on the schedule.

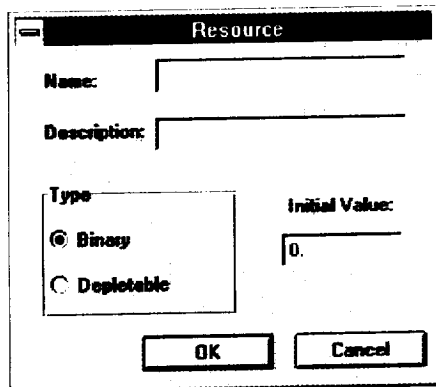
## Guess User's Guide

Resources may be added to the schedule by either selecting “New Resource” from the “Schedule” menu bar. Alternatively, the right mouse button may be clicked over the resource pane to bring up a popup menu. This brings up a dialog for adding a new resource (see figure 18).



**Figure 17 -  
Resource  
Menu**

The dialog is the same for both adding a new resource and editing an existing resource. Resources may be of two types - *binary* and *depletable*. Binary resources represent a fixed resource which may be utilized but once the utilization is finished the resource is available to the same extent that it was before being utilized. Typical examples of this might be scheduling personnel in performing a given task. The personnel would be utilized on a particular task; once the task was complete then they would be fully available.



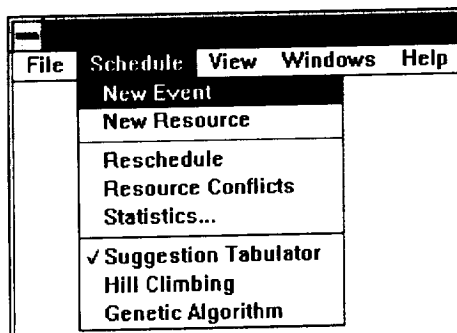
**Figure 18 - Resource Dialog**

Depletable resources, by contrast, are used up by the task and must be explicitly replenished. The usage specified in the dialog box is their initial starting amount. Events are assumed to consume a resource at a uniform rate during the event duration. Replenishment of the resource can be indicated by entering a negative value for resource usage in the resource constraint dialog (see editing resource constraints).

A resource cannot be changed from binary to depletable or vice-versa once it has been created. A resource may be deleted by clicking on the resource

in the resource graph with the right mouse button. This brings up a popup menu (see figure 17) with delete as one of the choices. Guess will then ask for confirmation of the deletion and if an affirmative answer is given will proceed to delete the resource.

### 6.6 Scheduling



**Figure 19 - Scheduling Methods**

Guess supports three different algorithms for scheduling events. They are the suggestion tabulator, hill climbing and genetic algorithm approaches. The suggestion tabulator is a linear approach, is very fast and works well for many types of problems where the constraint structure is not too complex. Hill climbing is a more complex algorithm and can provide solutions in some instances where the suggestion tabulator cannot provide very satisfactory solutions. Genetic algorithm tends to be time consuming but because

## Guess User's Guide

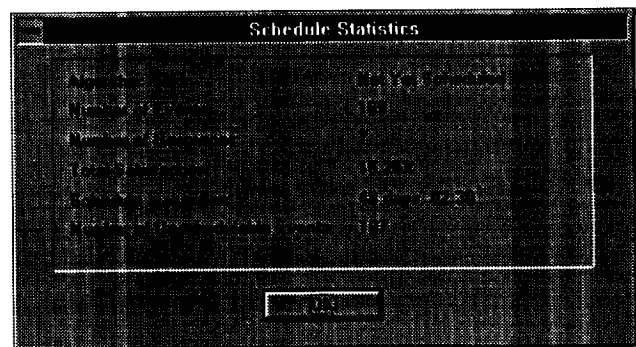
of its highly non-linear approach and probabilistic methods can often come up with novel solutions to the scheduling problem.

The choice of methods is made through the "Schedule" menu (see figure 19). The available solution techniques are shown on the bottom of the menu. The currently selected algorithm is shown with a check. By default, suggestion tabulator is selected. The method desired is changed by selecting the menu item associated with the algorithm desired.

Guess will reschedule the events according to the algorithm selected when the reschedule menu item is selected. The wait cursor will appear as Guess recalculates the schedule. The schedule views will then redraw reflecting the new schedule. Guess will reschedule events even though the resultant schedule may have some unscheduled events. These events will be rescheduled to make them as satisfied as the Guess scheduling engine can. The rescheduled event may however still be unschedulable. In that case the user has recourse to either try another scheduling algorithm which may give better results under the circumstances, examining the conflicts and modifying the constraints to allow the schedule to be satisfied or manually reschedule the event by dragging the gantt bars on the gantt chart display.

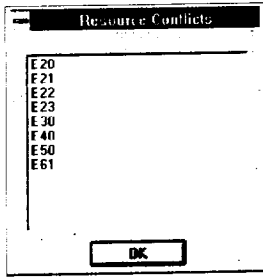
Rescheduling takes the latest calculated data as a starting point. This allows one to use different scheduling methods for stepwise refinement. One might start out using the suggestion tabulator and then refine the results obtained by choosing genetic algorithm as the scheduling method next. To compare schedules produced by different algorithms, the user should load a schedule using the "File Open" command, select the first scheduling method to be used and then click on "Reschedule". The next step is to use "Save As" to save the results to another file. Then the original data is loaded in another window using "File Open", another algorithm selected, and reschedule clicked. The user can then compare the resultant schedules.

Choosing "Statistics ..." from the Schedule menu brings up a statistics dialog allowing the user to check on the effectiveness of the resulting schedule. The total satisfaction is the actual satisfaction of the calculated schedule as a percentage of the maximum theoretical schedule satisfaction. The schedule makespan is the time interval from the time the first event starts to the time that the last event finishes. The number of unschedulable events is the number of events that could not be scheduled due to resource constraints being violated.



**Figure 20 - Statistics Dialog**

## Guess User's Guide



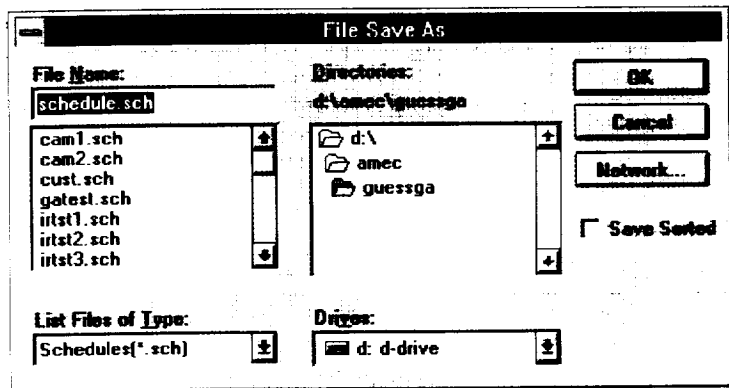
**Figure 21 -  
Resource Conflicts**

If there are resource conflicts, the “Resource Conflicts” menu item in the Schedule menu appears enabled. Selecting this item brings up a dialog showing a list of events that are subject to resource constraints. This allows the user to easily see which events have resource conflicts at a glance.

The events having resource constraints are also highlighted on the gantt chart in red. Individual satisfaction levels for each event are shown in the list pane located to the left of the gantt chart.

### 6.7 Saving Your Work

If you close a schedule and have made any changes to it, Guess always prompts you to save it. In addition, you can select either “Save” or “Save As” from the File menu. “Save” will save your work in the same file that you originally opened. If you have not previously saved the file then the File Save dialog box will appear. This dialog will also appear if you select “Save As”. A file can be selected using this dialog box in the same manner as the file open dialog. Normally Guess saves the events unsorted in the order in which they were entered. Checking the “Save Sorted” box overrides this behavior and forces Guess to save the events in whatever order was selected in the “View” menu.



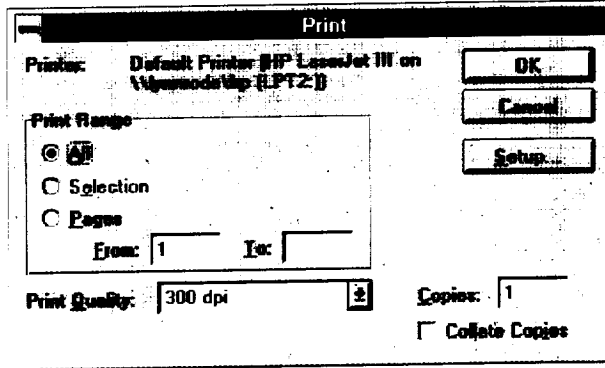
**Figure 22 - File Save As Dialog**

### 6.8 Printing the results

Choosing “Print Setup” from the File menu allows printer, paper orientation and size to be selected. Guess provides a print preview mode also available from the File menu that allows previewing the document on the screen as it would look on the printer prior to making a hard copy.



## Guess User's Guide



Printing is done by selecting the “Print” menu item from the File menu. This brings up a dialog allowing selection of the number of copies and the printer resolution. The user can choose to only print a section of the schedule by choosing the Pages button and entering a range of pages to be printed. Pressing OK causes the schedule to be printed.

Figure 23 - Print Dialog

### 7.0 Abbreviations And Acronyms

All abbreviations are defined when they first appear in the text. An alphabetized list of the definitions for abbreviations and acronyms used in this document is defined here.

AI	Artificial Intelligence
AMEC	American Minority Engineering Corporation
AMP	Automated Manifest Planner
API	Application Programming Interface
DDE	Dynamic Data Exchange
DLL	Dynamic Link Libraries
GUESS	Generically Used Expert Scheduling System
GUI	Graphical User Interface
IEEE	Institute for Electrical and Electronics Engineers
ISIS	International Society for Intelligent Systems
NASA	National Aeronautics and Space Administration
NASA-STD-2100-91	National Aeronautics and Space Administration Software Documentation Standard Software Engineering Program
ODBC	Open Database Connectivity
PARR	Planning and Resource Reasoning Shell
Sugtab	Suggestion Tabulator
SBIR	Small Business Innovative Research
VBX	Visual Basic Xtensions

### 8.0 Appendices

#### Appendix I

This appendix lists the function calls that a custom resource model library must support. It gives the function prototype, the arguments, and the value returned by the function. This does not inhibit the resource model by containing as many private functions as is necessary. Because the interface is defined in terms of a standard Windows API interface it does not presuppose the use of a specific language (e.g. C++) or a specific compiler vendor. It may be written using any tools that support writing 16 bit Windows 3.1 DLL's.

## Guess User's Guide

<b>long CreateResource (long ResourceID, FAR char * str)</b>	
ResourceID	Resource Identifier for use by GUESS. Resource ID is used in to get resource values during specific time periods from GUESS.
str	String identifying additional parameters required to properly initialize the saved resource. This string is the same string that is output by the WriteResource function.
Returns	An identifier with significance to the library. Identifier is opaque to GUESS and usually points to the structure representing the resource within the library.

<b>void DeleteResource (long ResourceID)</b>	
ResourceID	Resource identifier returned by library from a call to either <i>CreateResource</i> or <i>CreateNewResource</i> .

<b>double GetResourceValue (long id, long StartTime, long EndTime, double InitialValue, long Step, ResourceFunction Usage)</b>	
id	The id returned from <i>CreateResource</i> or <i>CreateNewResource</i> .
StartTime	Starting time for interval that resource value is being queried.
EndTime	Ending time for interval that resource value is being queried.
InitialValue	The initial value of the resource at the StartTime.
Step	This is the step time to be used. This allows <i>GUESS</i> to determine a step time which balances computational accuracy with speed.
Usage	A Usage function that the library iteratively calls to determine resource usage for each time slice within the time range sought. See <i>ResourceFunction</i> description for parameters.
Returns	Returns the value of the resource at the EndTime specified.

<b>double ( * ResourceFunction)(long id, long StartTime, long EndTime)</b>	
id	The resource id passed in as the ResourceID argument to either <i>CreateResource</i> or <i>CreateNewResource</i> .
StartTime	The starting time for collecting the resource usage. Typically the library calls this function repeatedly with StartTime incrementing by StepTime each time.
EndTime	This is the ending time for the time slice over which the resource value is desired.

## Guess User's Guide

### **double ( \* ResourceFunction)(long id, long StartTime, long EndTime)**

Returns	The resource usage over the specified time interval. <i>GUESS</i> calculates by figuring which events occur during this interval and what use they make of this resource.
---------	---

### **void WriteResource (long id, FAR char \*str, int maxlen)**

id	The resource id returned by either <i>CreateResource</i> or <i>CreateNewResource</i> .
str	A string in which to write the character string. The library is responsible for the format of this string. It is designed for performing file saves and will be read in again by <i>CreateResource</i> when the file is re-opened.
maxlen	The maximum length of the str string. The library is responsible to assure that the size of the string including the terminating NULL does not exceed this length.

### **FAR char \* GetName()**

Returns	Character string giving a human readable name for the custom resource model.
---------	--

### **FAR char \* GetDescription()**

Returns	Character string giving a description of the resource model.
---------	--



## Report Documentation Page

1. Report No. User's Guide-GUESS	2. Government Accession No.	3. Recipient's Catalog No.	
4. Title and Subtitle GUESS User's Guide		5. Report Date March 31, 1996	6. Performing Organization Code
		8. Performing Organization Report No.	
7. Author(s) Jay Liebowitz, Vijaya Krishnamurthy, Ira Rodens		10. Work Unit No.	
		11. Contract or Grant No. NAS5-38062	
9. Performing Organization Name and Address American Minority Engineering Corporation (AMEC) 10422 Armory Avenue, P.O. Box 509 Kensington, Maryland 20895		13. Type of Report and Period Covered Final; 4/194-3/31/96	
12. Sponsoring Agency Name and Address NASA Goddard Space Flight Center Greenbelt, Maryland 20771		14. Sponsoring Agency Code	
15. Supplementary Notes			
16. Abstract  This is the Version 1.0 of the GUESS User's Guide (Generically Used Expert Scheduling System).			
17. Key Words (Suggested by Author(s)) expert systems, scheduling, generic scheduling systems, intelligent scheduling		18. Distribution Statement  Unclassified-Limited	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 33	22. Price